School of Data and Computer Science, Sun Yat-sen University

E13 BP Algorithm (C++/Python)

16337102 Zilin Huang

December 6, 2018

Contents

1 Horse Colic Data Set

The description of the horse colic data set (`http://archive.ics.uci.edu/ml/datasets/Horse+Colic`) is as follows:

We aim at trying to predict if a horse with colic will live or die.

Note that we should deal with missing values in the data! Here are some options:

- Use the features mean value from all the available data.
- Fill in the unknown with a special value like -1.
- Ignore the instance.
- Use a mean value from similar items.
- Use another machine learning algorithm to predict the value.

2 Reference Materials

1. Stanford: **CS231n: Convolutional Neural Networks for Visual Recognition** by Fei-Fei Li,etc.
   - Course website: `http://cs231n.stanford.edu/2017/syllabus.html`
   - Video website: `https://www.bilibili.com/video/av17204303/?p=9&tdsourcetag=s_pctim_aiomsg`
2. **Machine Learning** by Hung-yi Lee
   - Course website: `http://speech.ee.ntu.edu.tw/~tlkagk/index.html`
   - Video website: `https://www.bilibili.com/video/av9770302/from=search`
3. A Simple neural network code template

```
# -*- coding: utf-8 -*
import random
import math

# Shorthand:
# "pd_" as a variable prefix means "partial derivative"
# "d_" as a variable prefix means "derivative"
# "_wrt_" is shorthand for "with respect to"
# "w_ho" and "w_ih" are the index of weights from hidden to output layer neurons and input to hidden layer neurons respectively

class NeuralNetwork:
```

```python
    LEARNING_RATE = 0.5
    def __init__(self, num_inputs, num_hidden, num_outputs, hidden_layer_weights = None, hidden_layer_bias = None, output_layer_weights = None, output_layer_bias = None):
    #Your Code Here

    def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights):
    #Your Code Here

    def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(self, output_layer_weights):
    #Your Code Here

    def inspect(self):
        print('------')
        print('* Inputs: {}'.format(self.num_inputs))
        print('------')
        print('Hidden Layer')
        self.hidden_layer.inspect()
        print('------')
        print('* Output Layer')
        self.output_layer.inspect()
        print('------')

    def feed_forward(self, inputs):
        #Your Code Here

    # Uses online learning, ie updating the weights after each training case
    def train(self, training_inputs, training_outputs):
        self.feed_forward(training_inputs)

        # 1. Output neuron deltas
        #Your Code Here
        #  E/  z

        # 2. Hidden neuron deltas
        # We need to calculate the derivative of the error with respect to the output of each hidden layer neuron
        # dE/dy =   E/  z  *  z/  y  =   E/  z  * w
        #  E/  z  = dE/dy  *  z  /
```

```
        #Your Code Here

        # 3. Update output neuron weights
        #   E  /  w     =  E/  z   *   z  /  w
        #  w =   *   E  /  w
        #Your Code Here

        # 4. Update hidden neuron weights
        #   E  /  w   =  E/  z   *   z  /  w
        #  w =   *   E  /  w
        #Your Code Here

    def calculate_total_error(self, training_sets):
        #Your Code Here
        return total_error

class NeuronLayer:
    def __init__(self, num_neurons, bias):

        # Every neuron in a layer shares the same bias
        self.bias = bias if bias else random.random()

        self.neurons = []
        for i in range(num_neurons):
            self.neurons.append(Neuron(self.bias))

    def inspect(self):
        print('Neurons:', len(self.neurons))
        for n in range(len(self.neurons)):
            print('␣Neuron', n)
            for w in range(len(self.neurons[n].weights)):
                print('␣␣Weight:', self.neurons[n].weights[w])
            print('␣␣Bias:', self.bias)

    def feed_forward(self, inputs):
        outputs = []
        for neuron in self.neurons:
            outputs.append(neuron.calculate_output(inputs))
        return outputs

    def get_outputs(self):
```

```
        outputs = []
        for neuron in self.neurons:
            outputs.append(neuron.output)
        return outputs

class Neuron:
    def __init__(self, bias):
        self.bias = bias
        self.weights = []

    def calculate_output(self, inputs):
    #Your Code Here

    def calculate_total_net_input(self):
    #Your Code Here

    # Apply the logistic function to squash the output of the neuron
    # The result is sometimes referred to as 'net' [2] or 'net' [1]
    def squash(self, total_net_input):
    #Your Code Here

    # Determine how much the neuron's total input has to change to move closer to the expected output
    #
    # Now that we have the partial derivative of the error with respect to the output (  E/  y  ) and
    # the derivative of the output with respect to the total net input (dy  / dz  ) we can calculate
    # the partial derivative of the error with respect to the total net input.
    # This value is also known as the delta (  ) [1]
    #   =  E/  z  =  E/  y  * dy  /dz
    #
    def calculate_pd_error_wrt_total_net_input(self, target_output):
    #Your Code Here

    # The error for each neuron is calculated by the Mean Square Error method:
    def calculate_error(self, target_output):
    #Your Code Here

    # The partial derivate of the error with respect to actual output then is calculated by:
```

```
    # = 2 * 0.5 * (target output - actual output) ^ (2 - 1) * -1
    # = -(target output - actual output)
    #
    # The Wikipedia article on backpropagation [1] simplifies to the following, but most other learning material does not [2]
    # = actual output - target output
    #
    # Alternative, you can use (target - output), but then need to add it during backpropagation [3]
    #
    # Note that the actual output of the output neuron is often written as y and target output as t so:
    # = E/ y = -(t - y )
    def calculate_pd_error_wrt_output(self, target_output):
    #Your Code Here

    # The total net input into the neuron is squashed using logistic function to calculate the neuron's output:
    # y = = 1 / (1 + e^(-z ))
    # Note that where represents the output of the neurons in whatever layer we're looking at and represents the layer below it
    #
    # The derivative (not partial derivative since there is only one variable) of the output then is:
    # dy /dz = y * (1 - y )
    def calculate_pd_total_net_input_wrt_input(self):
    #Your Code Here

    # The total net input is the weighted sum of all the inputs to the neuron and their respective weights:
    # = z = net = x w + x w ...
    #
    # The partial derivative of the total net input with respective to a given weight (with everything else held constant) then is:
    # = z / w = some constant + 1 * x w ^(1-0) + some constant ... = x
    def calculate_pd_total_net_input_wrt_weight(self, index):
    #Your Code Here

# An example:
```

```
nn = NeuralNetwork(2, 2, 2, hidden_layer_weights=[0.15, 0.2, 0.25, 0.3],
    hidden_layer_bias=0.35, output_layer_weights=[0.4, 0.45, 0.5, 0.55],
    output_layer_bias=0.6)
for i in range(10000):
    nn.train([0.05, 0.1], [0.01, 0.99])
    print(i, round(nn.calculate_total_error([[[0.05, 0.1], [0.01, 0.99]]]), 9))
```

3 Tasks

- Given the training set `horse-colic.data` and the testing set `horse-colic.test`, implement the BP algorithm and establish a neural network to predict if horses with colic will live or die. In addition, you should calculate the accuracy rate.
- Please submit a file named `E13_YourNumber.pdf` and send it to `ai_2018@foxmail.com`

4 Codes and Results

4.1 Codes

exp.py

```
import random
import math
import numpy as np
import pandas as pd
import copy

class NeuralNetwork:
    LEARNING_RATE = 0.5

    def __init__(self, num_inputs, num_hidden, num_outputs, hidden_layer_weights=None,
        hidden_layer_bias=None, output_layer_weights=None, output_layer_bias=None):
        #Your Code Here
        self.num_in = num_inputs
        self.num_hid = num_hidden
        self.num_out = num_outputs
        self.hidden_layer = NeuronLayer(num_hidden, hidden_layer_bias)
        self.output_layer = NeuronLayer(num_outputs, output_layer_bias)
        self.init_weights_from_inputs_to_hidden_layer_neurons(hidden_layer_weights)
```

```
        self.init_weights_from_hidden_layer_neurons_to_output_layer_neurons(output_layer_weights)

    #
    def init_weights_from_inputs_to_hidden_layer_neurons(self, hidden_layer_weights):
        if hidden_layer_weights:
            counter = 0
            for i in range(self.num_hid):
                self.hidden_layer.neurons[i].weights = hidden_layer_weights[counter:counter + self.num_in - 1]
                counter += self.num_in
        else:
            for i in range(self.num_hid):
                weights = [random.random() for i in range(self.num_in)]
                self.hidden_layer.neurons[i].weights = copy.deepcopy(weights)

    def init_weights_from_hidden_layer_neurons_to_output_layer_neurons(self, output_layer_weights):
        if output_layer_weights:
            counter = 0
            for i in range(self.num_out):
                self.output_layer.neurons[i].weights = output_layer_weights[counter:counter + self.num_hid - 1]
                counter += self.num_hid
        else:
            for i in range(self.num_out):
                weights = [random.random() for i in range(self.num_hid)]
                self.output_layer.neurons[i].weights = copy.deepcopy(weights)

    def inspect(self):
        print('------')
        print('* Inputs: {}'.format(self.num_in))
        print('------')
        print('Hidden Layer')
        self.hidden_layer.inspect()
        print('------')
        print('* Output Layer')
        self.output_layer.inspect()
        print('------')

    def feed_forward(self, inputs):
```

```
        mids = self.hidden_layer.feed_forward(inputs)
        self.output_layer.feed_forward(mids)
        total = 0
        for i in range(self.num_out):
            total += self.output_layer.neurons[i].output
        for i in range(self.num_out):
            self.output_layer.neurons[i].output /= total

    def train(self, training_inputs, training_outputs):
        self.feed_forward(training_inputs)

        deltas_out = []
        for i in range(self.num_out):
            delta = self.hidden_layer.neurons[i].calculate_pd_error_wrt_total_net_input(int(training_outputs[i]))
            deltas_out.append(delta)

        deltas_hid = []
        for i in range(self.num_hid):
            dE_dyi = 0
            for j in range(self.num_out):
                dE_dyi += self.output_layer.neurons[j].weights[i] * deltas_out[j]
            a = self.hidden_layer.neurons[i].output
            delta = a * (1 - a) * dE_dyi
            deltas_hid.append(delta)

        w_out = [ [] for i in range(self.num_out) ]
        for i in range(self.num_out):
            for j in range(self.num_hid):
                w = self.LEARNING_RATE * \
                    deltas_out[i] * \
                    self.hidden_layer.neurons[j].output
                w_out[i].append(w)
        for i in range(self.num_out):
            for j in range(self.num_hid):
                self.output_layer.neurons[i].weights[j] += w_out[i][j]

        w_hid = [[] for i in range(self.num_hid)]
        for i in range(self.num_hid):
            for j in range(self.num_in):
                w = self.LEARNING_RATE * \
```

```
                    deltas_hid[i] * \
                    self.hidden_layer.neurons[i].calculate_pd_total_net_input_wrt_weight(j)
                w_hid[i].append(w)
        for i in range(self.num_hid):
            for j in range(self.num_hid):
                self.hidden_layer.neurons[i].weights[j] += w_hid[i][j]


class NeuronLayer:
    def __init__(self, num_neurons, bias):

        # Every neuron in a layer shares the same bias
        self.bias = bias if bias else random.random()

        self.neurons = []
        for i in range(num_neurons):
            self.neurons.append(Neuron(self.bias))

    def inspect(self):
        print('Neurons:', len(self.neurons))
        for n in range(len(self.neurons)):
            print(' Neuron', n)
            for w in range(len(self.neurons[n].weights)):
                print('  Weight:', self.neurons[n].weights[w])
            print('  Bias:', self.bias)

    def feed_forward(self, inputs):
        outputs = []
        for neuron in self.neurons:
            outputs.append(neuron.calculate_output(inputs))
        return outputs

    def get_outputs(self):
        outputs = []
        for neuron in self.neurons:
            outputs.append(neuron.output)
        return outputs

class Neuron:
```

```python
    def __init__(self, bias):
        self.bias = bias
        self.weights = []
        self.weighted_input = 0
        self.output = 0
        self.inputs = []

    #
    def calculate_output(self, inputs):
        self.weighted_input = self.calculate_total_net_input(inputs)
        output = self.squash(self.weighted_input)
        self.output = output
        return output

    def calculate_total_net_input(self, inputs):
        total_net_input = 0
        self.inputs = copy.deepcopy(inputs)
        num = len(self.weights)
        for i in range(num):
            total_net_input += self.weights[i] * inputs[i]
        total_net_input += self.bias
        return total_net_input

    def squash(self, total_net_input):
        down = 1 + pow(np.e, -total_net_input)
        output = 1 / down
        return output

    def calculate_pd_error_wrt_total_net_input(self, target_output):
        delta = self.calculate_pd_error_wrt_output(
            target_output) * self.calculate_pd_total_net_input_wrt_input()
        return delta

    def calculate_error(self, target_output):
        #Your Code Here
        error = pow(self.output - target_output, 2) * 0.5
        return error

    def calculate_pd_error_wrt_output(self, target_output):
        pd_error = (target_output - self.output)
        return pd_error
```

```python

    def calculate_pd_total_net_input_wrt_input(self):
        pd = self.output * (1 - self.output)
        return pd
x
    def calculate_pd_total_net_input_wrt_weight(self, index):
        x_index = self.inputs[index]
        return x_index

def test(network, data, result):
    global result_dic
    num = len(data)
    correct = 0
    outputs = []
    for i in range(num):
        inputs = data.iloc[i].tolist()
        network.feed_forward(inputs)
        max_output = 0
        position = -1
        for j in range(network.num_out):
            if max_output < network.output_layer.neurons[j].output:
                max_output = network.output_layer.neurons[j].output
                position = j
        outputs.append(result_dic[position])
    for i in range(num):
        if outputs[i] == result.iloc[i]:
            correct += 1
    return correct / num

result_dic = {0:'100', 1:'010', 2:'001'}
data = pd.read_csv('train.data', header=None)
result = data[22]
result = result.replace(
    {-1:'100', 1:'100', 2:'010', 3:'001'})
data = data / 10
#
del data[2]
del data[22]
del data[25]
del data[26]
del data[27]
```

```python

test_data = pd.read_csv('train.data', header=None)
test_result = test_data[22]
test_result = test_result.replace(
    {-1: '100', 1: '100', 2: '010', 3: '001'})
test_data = test_data / 10
del test_data[2]
del test_data[22]
del test_data[25]
del test_data[26]
del test_data[27]

nn = NeuralNetwork(22, 6, 3, hidden_layer_weights=None,
                   hidden_layer_bias=random.random(),
                   output_layer_weights=None,
                   output_layer_bias=random.random())
num = len(data)
for j in range(1, 121):
    accuracy = 0
    if j % 8 == 0:
        print('after training ' + str(j) + ' times, accuracy:')
        accuracy = test(nn, test_data, test_result)
        print(accuracy)
        if accuracy >= 0.6:
            nn.hidden_layer.inspect()
            nn.output_layer.inspect()
    for i in range(num):
        nn.train(data.iloc[i].tolist(), result.iloc[i])
```

deal$_d$*ata.py*

```python
def deal_data(input_name, output_name):
    train_data = open(input_name)
    f = open(output_name, 'w+')
    data = []
    while 1:
        line_ = train_data.readline()
        if not line_:
            break
        line_ = line_[:-1]
        line_ = line_.split(' ')
        data.append(line_)
```

```python
    num = len(data)
    for i in range(num):
        attr_num = len(data[i])
        no = False
        for j in range(attr_num):
            if data[i][j] == '?':
                data[i][j] = '-1'
            if j == 28:
                no = True
        if no:
            data[i] = data[i][:-1]
        print(','.join(data[i]), file=f)
    train_data.close()
    f.close()


deal_data('horse-colic.data', 'train.data')
deal_data('horse-colic.test', 'test.data')
```

4.2 Result

```
after training 8 times, accuracy:
0.25666666666666665
```

```
after training 120 times, accuracy:
0.5966666666666667
```